Enhanced Sampling Toolkit Documentation

Release 1.0

Jeremy Tempkin

April 07, 2015

CONTENTS

Contents:

CHAPTER

ONE

WALKER API

Created on Fri Jun 6 15:16:18 2014

NOTE: we may want to use abstract properties in the future.

@author: jeremytempkin

class walker.**velocityWalker**

This is an abstract class for dynamics which have a velocity component, such as a protein under Langevin dynamics. It extends the walker abstract class by adding additional abstract methods that are necessary for a walker that has a velocity.

drawVel()

Draws a new value of the velocities for the walker.

reverseVel()

This function reverses the velocity of the walker.

class walker.**walker**

Defines an abstract walker object. It functions as an interface, providing methods that other walkers should implement.

close()

Destroys the walker, and does some housecleaning.

equilibrate(*colvar*)

This function sets the walker at a specific configuration in the collective variable space. At minimum, any implementation will need to give it the configuration.

getColvars()

This function returns the location of the walker in the collective variable space.

getConfig()

This function should return the configuration of the walker in configurtion space.

propagate(*numsteps*)

This function propagates the simulation forward a given number of steps. It takes as an argument at least numsteps, the number of steps to progagate forward.

setConfig(*configuration*)

This function should set the system at a specific place in configurtin space. At minimum, it should take some sort of specification of the configuration.

CHAPTER

TWO

LAMMPS WALKER MODULE

This file implements the LAMMPS walker abstraction layer. The core of this idea is that

class lammpsWalker.**lammpsWalker**(*inputFilename*, *logFilename*, *index=0*, *debug=False*)

This class implements the enhanced sampling walker API for the bindings to the LAMMPS package. To check the math formatting, here is an example $e^{ipi} = -1$.

Some usage issues to note:

1) Collective variables (CVs) are defined to the walker by constructing a list of CVs internally in the walker.colvars object. These CVs list takes the following format:

["coordinateType", atomids....]

The coordinate type specifies which type of coordinate the CV is (i.e. bond, angle, dihedral, etc.). The next items in the list are the atom indecies involved in this specific instance of the CV.

The walker will use this list to initialize them to the underlying LAMMPS objects.

close()

This function closes the LAMMPS object.

command(*command*)

This function allows the user to issue a LAMMPS command directly to the LAMMPS object.

destroyColvars()

This function removes the colvars set by initColVars().

drawVel(*distType='gaussian'*, *temperature=310.0*)

This function redraws the velocities from a maxwell-boltzmann dist.

equilibrate(*center*, *restraint*, *numSteps*)

This function prepares a LAMMPS image to be at the specified target position given by the vector 'center' passed and an arguments.

getColvars()

This function returns the current position of the LAMMPS simulation in colvars space.

getConfig()

This function returns the current position of the LAMMPS simulation.

getVel()

This function returns the current velocities from the LAMMPS simulation.

minimize(*args=None*)

This function runs a minimization routine with the specified type.

propagate(*numSteps*, *pre='no'*, *post='no'*)

This function issues a run command to the underlying dynamics to propagate the dynamics a given number of steps.

reverseVel()
This function reverses the velocities of a given LAMMPS simulation

setColvars()
This function initializes the collective variable for a LAMMPS simulation that is handed to this object.

Currently supports the following cv's:

- bond
- angle
- dihedral
- x, y or z position coordinates
- x, y or z velocity components

These are parsed and sent to the underlying LAMMPS object directly using the LAMMPS syntax for these variable.

The implementation first creates a labeled group in LAMMPS containing the atoms used in the CV. Then a compute is initialized using that group.

setConfig(*config*)
This routine sets the internal configuration.

setDynamics()
This routine sets the dynamics for the walker.

setTemperature(*temp*)
This function sets the temperature of the walker object.

NOTE THAT THIS DOES NOT ALTER THE DYNAMICS THERMOSTAT. LAMMPS REQUIRES RESETING THIS THERMOSTAT. WE WILL LOOK INTO HOW TO DO THIS.

setTimestep(*timestep*)
This routine sets the dynamics time step.

setVel(*vel*)
This function sets the velocity to the lammps simulation.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

l

w